Scenario Based Jenkins Implementations

1. Handle Long-Running Jobs with Timeouts

Scenario: Automatically terminate a job if it exceeds a certain duration.
Code Explanation:
This pipeline ensures that a stage does not run indefinitely by specifying a timeout duration. In this example, the timeout block is used in the Build stage to limit execution to 10 minutes. If the task exceeds this duration, Jenkins terminates it.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        timeout(time: 10, unit: 'MINUTES') {
          echo 'Building project...'
          sh 'sleep 600' // Simulate a long-running task
        }
      }
    }
  }
}
```

---

2. Conditional Stage Execution Based on File Change

Scenario: Execute specific stages only when particular files are modified in the repository.
Code Explanation:
The when block is used to check if certain files (e.g., *.test.js) have been modified. If the condition is true, the Run Tests stage is executed.

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        git branch: 'main', url: 'https://github.com/your-repo/your-project.git'
      }
    }
    stage('Run Tests') {
      when {
        changeset "**/*.test.js" // Trigger stage if test files are modified
      }
      steps {
        echo 'Running tests...'
        sh 'npm test'
      }
    }
  }
}
```

---

3. Post-Build Cleanup

Scenario: Clean up workspace after a build to free up disk space.
Code Explanation:
The post section ensures the workspace is cleaned after the build, regardless of success or failure. The cleanWs() step removes all files in the workspace.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building project...'
        // Build logic
      }
    }
  }
  post {
    always {
      echo 'Cleaning up workspace...'
      cleanWs()
    }
  }
}
```

---

4. Retry Failed Steps

Scenario: Automatically retry a failing step up to a specified number of attempts.
Code Explanation:
The retry block wraps a command, allowing Jenkins to retry it up to 3 times if it fails. In this example, exit 1 simulates a failure.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        retry(3) {
          echo 'Attempting to build...'
          sh 'exit 1' // Simulate a failure
        }
      }
    }
  }
}
```

---

5. Multi-Environment Deployment Using Parameters

Scenario: Deploy an application to different environments (Dev, QA, Prod) based on a parameter selected during the build.
Code Explanation:

The parameters block defines a dropdown to select the environment. The deployment logic is controlled by a script block that uses if-else conditions to deploy to the selected environment.

```
pipeline {
  agent any
  parameters {
    choice(name: 'ENV', choices: ['Dev', 'QA', 'Prod'], description: 'Select the environment to
deploy')
  }
  stages {
    stage('Build') {
      steps {
        echo "Building the application for environment: ${params.ENV}"
      }
    }
    stage('Deploy') {
      steps {
        script {
          if (params.ENV == 'Dev') {
            echo 'Deploying to Dev environment...'
          } else if (params.ENV == 'QA') {
            echo 'Deploying to QA environment...'
          } else if (params.ENV == 'Prod') {
            echo 'Deploying to Prod environment...'
          }
        }
      }
    }
  }
}
```

---

6. Parallel Testing

Scenario: Run multiple test suites (e.g., unit tests, integration tests, UI tests) in parallel to speed up the process.
Code Explanation:
The parallel block allows the execution of multiple stages concurrently, reducing total runtime.

```
pipeline {
  agent any
  stages {
    stage('Parallel Testing') {
      parallel {
        stage('Unit Tests') {
          steps {
            echo 'Running Unit Tests...'
          }
        }
        stage('Integration Tests') {
          steps {
            echo 'Running Integration Tests...'
```

```
                    }
                }
                stage('UI Tests') {
                    steps {
                        echo 'Running UI Tests...'
                    }
                }
            }
        }
    }
}
```

---

7. Conditional Stages Based on Branch

Scenario: Execute specific stages only when a certain branch is being built.
Code Explanation:
The when block checks the branch name and executes the respective stage if the condition is met.

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: '*/main', url: 'https://github.com/your-repo/your-project.git'
            }
        }
        stage('Build') {
            when {
                branch 'main'
            }
            steps {
                echo 'Building for the main branch...'
            }
        }
        stage('Test') {
            when {
                branch 'feature/*'
            }
            steps {
                echo 'Testing for a feature branch...'
            }
        }
    }
}
```

---

8. Archive and Publish Build Artifacts

Scenario: Save build artifacts and make them available for download.
Code Explanation:
The archiveArtifacts step saves specified files (e.g., build-artifact.zip) and fingerprints them for traceability.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building the application...'
                sh 'touch build-artifact.zip'
            }
        }
        stage('Archive Artifacts') {
            steps {
                archiveArtifacts artifacts: 'build-artifact.zip', fingerprint: true
            }
        }
    }
}
```